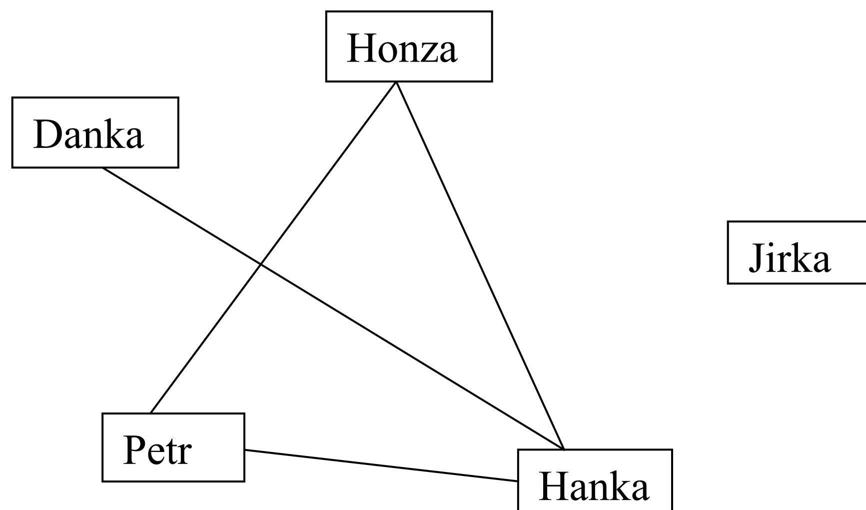
ADT Graf

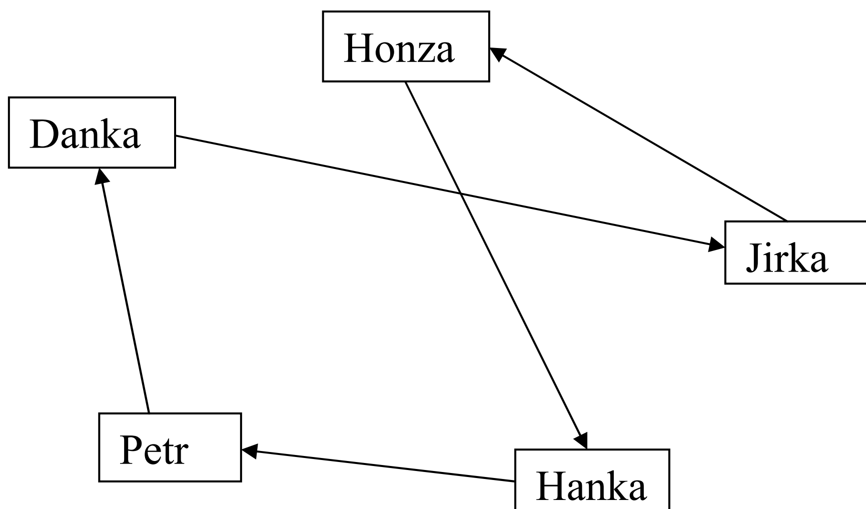
co je to ?

model (abstrakce) vztahů mezi prvky

vztah: mají se rádi *(vztahy navrhli studenti na přednášce)*



vztah: má rád(a) *(vztahy navrhli studenti na přednášce)*



vztah: mezi městy je přímé spojení a rychlost

Zdroj: www.cesnet.cz

prvek - vrchol

vztah - hrana

Formálně

$G = (V, H)$

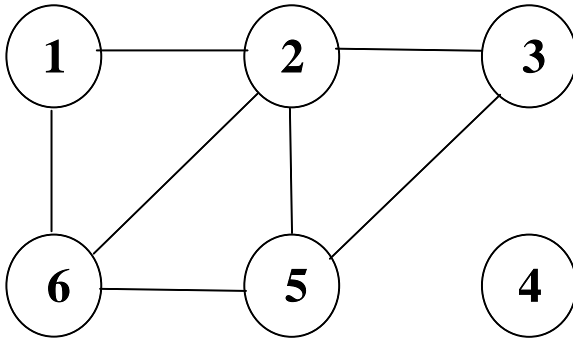
V – množina vrcholů (uzlů) $|V|$ - počet vrcholů
H – množina hran $|H|$ - počet hran

hrana - dvojice (u,v) $u,v \in V$

orientovaná hrana $(u,v) \neq (v,u)$
začáteční a koncový vrchol

$V(G)$ – množina vcholů grafu G
$H(G)$ – množina hran grafu G

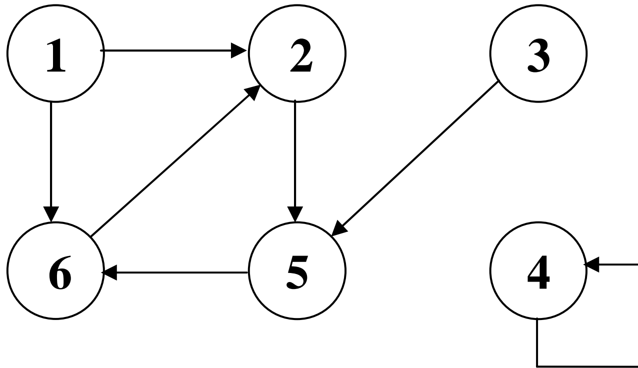
Příklad neorientovaného grafu



$V = \{ 1, 2, 3, 4, 5, 6 \}$ $|V| = 6$

$H = \{ (1,2), (1,6), (2,3), (2,5), (2,6), (3,5), (5,6) \}$ $|H| = 7$

Příklad orientovaného grafu



V = { 1, 2, 3, 4, 5, 6 } ***|V | = 6***

H = { (1,2) (1,6) (2,5) (3,5) (4,4) (5,6) (6,2) } ***|H | = 7***

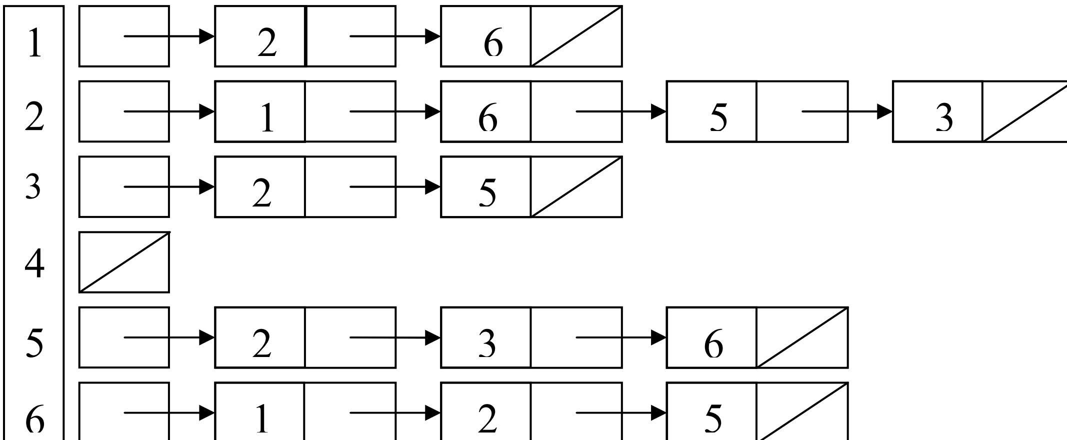
Reprezentace grafu v informatice

- seznamy sousednosti
- matice sousednosti
- pro orientované i neorientované grafy

Seznamy sousednosti

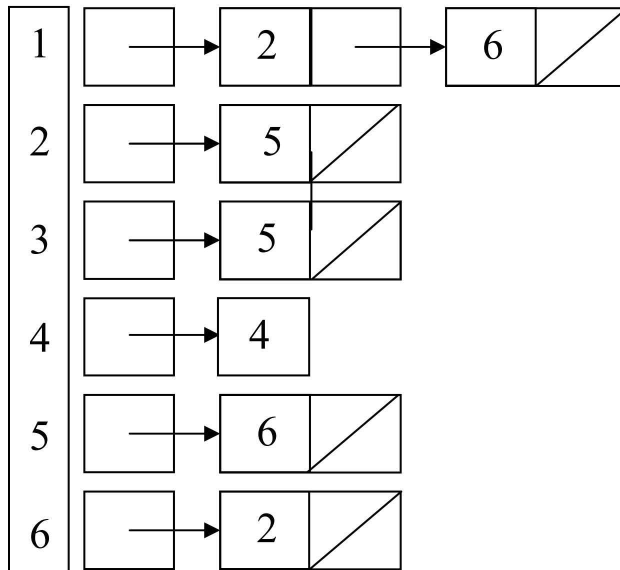
- pro každý vrchol je vytvořen *seznam* sousedů
- sousedi jsou uloženy libovolném pořadí

Reprezentace neorientovaného grafu



Celková délka seznamů sousednosti pro neorientovaný graf je *2|H|*.

Reprezentace orientovaného grafu



Celková délka seznamů sousednosti pro orientovaný graf je |*H*|.

Paměť

Pro neorientovaný i orientovaný graf požadovaná paměť
je $\Theta(|V| + |H|)$.

Implementace

Označme vrcholy čísly *0, …, |V|-1*.

vrchol:

```
class Vrchol {
  ... // další položky pro informace o vrcholu
  Soused sousedi;

  Vrchol () {
    ... // inicializace dalších položek
    sousedi = null;
  }
}
```

sousedi:

```
class Soused {
  int vrchol;
  ... // další položky pro informace o hraně
  Soused dalsi;

  Soused (int v) {
    ...// inicializace dalších položek
    vrchol = v;
    dalsi = null;
  }
}
```

graf = pole vrcholů se seznamy sousedů

```
Vrchol[] vrcholy = new Vrchol[|V|];
```

vložení hrany:

```
void hrana(int z, int kam) {
    Soused s = new Soused(kam);
    s.dalsi = vrcholy[z].sousedi;
    vrcholy[z].sousedi = s;
}
```

- reprezentace seznamem sousednosti je vhodná i pro **ohodnocené** grafy

- ohodnocení grafu *G*

$$w: H(G) \rightarrow \mathbf{R}$$

- ***w(u,v)*** se uloží ve vrcholu ***v*** seznamu sousedů vrcholu ***u***

zjišťování existence hrany ***(u,v)*** = hledání vrcholu ***v*** v seznamu sousedů vrcholu ***u*** – ***O(|V|)***

Matice sousednosti

Označme vrcholy čísly $1, \ldots, |V|$.

Matice sousednosti je matice $S = (s_{ij}),\ i,j = 1, \ldots, |V|$, přičemž

je-li $(i,j) \in H,\ s_{ij} = 1$
jinak $s_{ij} = 0$

Reprezentace neorientovaného grafu

	1	*2*	*3*	*4*	*5*	*6*
1	0	1	0	0	0	1
2	1	0	1	0	1	1
3	0	1	0	0	1	0
4	0	0	0	0	0	0
5	0	1	1	0	0	1
6	1	1	0	0	1	0

Reprezentace orientovaného grafu

	1	*2*	*3*	*4*	*5*	*6*
1	0	1	0	0	0	1
2	0	0	0	0	1	0
3	0	0	0	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	0	1
6	0	1	0	0	0	0

Paměť

Pro neorientovaný i orientovaný graf požadovaná paměť je $\Theta(|V|^2)$ (bez ohledu na počet hran).

graf = dvourozměrné pole

```
int[][] s = new int [|V|], [|V|];
```

Poznámky:

Indexy v Javě budou ***0*** až ***|V |-1***

Pro neorientovaný graf je $S = S^T$, kde S^T je transponovaná matice, což umožňuje snížit nároky na paměť téměř na polovinu.

Maticí sousednosti lze implementovat i ohodnocený graf. Pro ohodnocený graf je

$s_{uv} = w(u,v)$, je-li $(u,v) \in H(G)$
s_{uv} je hodnota mimo hodnot možných ohodnocení, je-li $(u,v) \notin H(G)$

V případě neohodnoceného grafu, možno prvky matice sousednosti uložit v bitech.

Jak zvolit reperezentaci?

Je-li $|H| << |V|^2$ graf se nazývá *řídký,*
obvykle je vhodnější použít seznam sousednosti.

Je-li $|H| \sim |V|^2$ graf se nazývá *hustý,*
obvykle je vhodnější použít matici sousednosti.

Matici sousednosti je vhodnější použít také, je-li nutno rychle zjistit existenci hrany.

PROHLEDÁVÁNÍ DO ŠÍŘKY
BREATH-FIRST SEARCH

Algoritmus

Z vybraného vrcholu ***s*** nalezneme všechny vrcholy ve vzdálenosti ***k*** od vrcholu *s* předtím, než nalezneme vrcholy ve vzdálenosti ***k+1****.*

Nalezený vrchol obarvíme šedě a uložíme ho do fronty, přičemž začneme vrcholem ***s****.*

Po nalezení všech sousedů vrchol obarvíme černě. Není-li fronta prázdná, pokračujeme dalším vrcholem z fronty.

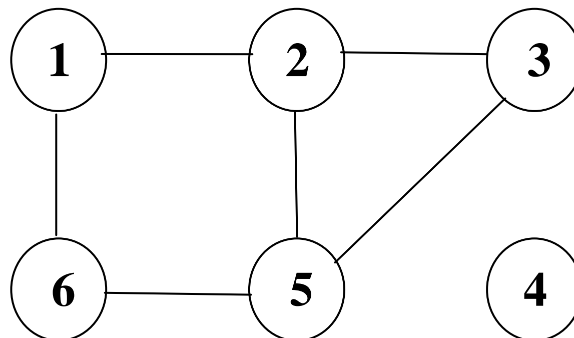
Implementace

Další informace v prvcích pole **`vrcholy[] třídy Vrchol`** jsou

`barva` – barva uzlu, na začátku bílá
`vzdalenost` – vzdálenost od uzlu ***s****,* na začátku ∞
`predchudce` – číslo předcházejícího uzlu, na začátku -1 = není žádný předchůdce

```
void BFS (int s) {
    IntFronta f = new IntFronta();
    vrcholy[s].barva = 'S';
    vrcholy[s].vzdalenost = 0;
    f.vloz(s);
    while(!f.jePrazdna()) {
      int u = f.vyber();
      for (všechny vrcholy v sousedící s u) {
        if (vrcholy[v].barva == 'B') {
          vrcholy[v].barva = 'S';
          vrcholy[v].vzdalenost =
                        vrcholy[u].vzdalenost+1;
          vrcholy[v].predchudce = u;
          f.vloz(v);
        }
      }
      vrcholy[u].barva = 'C';
    }
  }
```

Příklad



Složitost

Obílení všech vrcholů při inicializaci trvá $O(|V|)$ *(není v BFS).*

Po vložení do fronty, vrcholy již nikdy nejsou obíleny, a protože vložení a vybrání z fronty je $O(1)$, trvání těchto operací je $O(|V|)$.

Sousedé všech vrcholů jsou procházeni, jenom jednou, totiž když je vrchol vybrán z fronty, a jejich celková délka je $\Theta(|H|)$.

Procházení tedy trvá $O(|H|)$.

Čas BFS je tedy $O(|V| + |H|)$.

Vlastnosti BFS algoritmu

- nalezneme všechny dosažitelné vrcholy z vybraného vrcholu ***s***
- vypočteme vzdálenost (počet hran) k objeveným vrcholům od ***s***
- v grafu ***G(V,H)*** definujeme délku nejkratší cesty mezi vrcholy ***s,v∈V*** jako minimální počet hran všech cest z vrcholu ***s*** do vrcholu ***v***; délka nejkratší cesty mezi vrcholy ***s*** a ***v*** potom je rovná hloubce vrcholu ***v*** v BFS stromě s kořenem ***s***, která je uložená v položce **`vzdalenost`**
- vytvoříme BFS strom všech dosažitelných vrcholů, kterého kořen je ***s***

tisk vrcholů na nejkratší cestě z vrcholu *s* do vrcholu *v* po vykonání BFS

```
void tiskCesty(int s, int v) {
    if(v == s)
      System.out.println(s+" ");
    else {
      if(vrcholy[v].predchudce == -1)
        System.out.println("cesta neexistuje");
      else {
        tiskCesty(s,vrcholy[v].predchudce);
        System.out.println(v+" ");
      }
    }
  }
```

Poznámky:

BFS je analogie průchodu stromem po úrovních.

BFS je pro orientované i neorientované grafy.

BFS je základem dalších algoritmů (Primův algoritmus minimální kostry, Dijkstrův algoritmus minimální cesty).

PROHLEDÁVÁNÍ DO HLOUBKY
DEPTH-FIRST SEARCH

Hledáme, když je to možné, napřed do „hloubky", tj. do větší vzdálenosti a nalezené vrcholy obarvíme šedě

Po průchodu všemi hranami z vyšetřovaného vrcholu, vrchol obarvíme černě a vrátíme se k jeho předchůdci nebo skončíme

Kromě položek **barva** a **predchudce** zavedeme do dalších informací dvě časové značky (jednotka „času" = přechod na další vrchol)

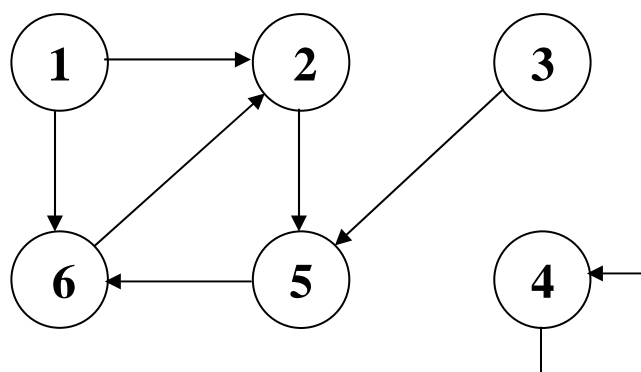
`objeven` – čas nalezení vrcholu, kdy je obarven šedě
`dokoncen` – čas konce procházení seznamu sousedů vrcholu, kdy je vrchol obarven černě

Hodnoty časových značek jsou mezi 1 a 2|H|.

Pro každý vrchol *u* platí *objeven(u)* < *dokoncen(u).*

```
void DFS (int u) {
    vrcholy[u].barva = 'S';
    cas = cas + 1;
    vrcholy[u].objeven = cas;
    for (všechny vrcholy v sousedící s u) {
      if (vrcholy[v].barva == 'B') {
        vrcholy[v].predchudce = u;
        DFS(v);
      }
    }
    vrcholy[u].barva = 'C';
    vrcholy[u].dokoncen = cas = cas + 1;
  }
```

Příklad



Složitost

V úvodu a závěru metody DFS je každý vrchol obarven napřed šedě a potom černě právě jednou, což trvá ***O(|V |).***

DFS je voláno pro bílé vrcholy, které jsou ihned zašeděny a v DFS se cyklus **for** vykoná pro všechny hrany z něho vycházející, což je celkem pro všechny vrcholy ***O(|H|)***.

Čas výpočtu DFS tedy je ***O(|V | + |H|).***

Vlastnosti

DFS nalezne pro začáteční vrchol DFS strom dosažitelných vrcholů.

Jestliže po vykonání DFS zůstaly neobjevené vrcholy, jeden vybereme a postup opakujeme, vznikne tak les DFS stromů.

DFS využívají jiné algoritmy.

Poznámky:

Analogie ~order průchodu stromem.

DFS je pro orientované i neorientované grafy.

Rekurzivní volání lze odstranit použitím zásobníku.

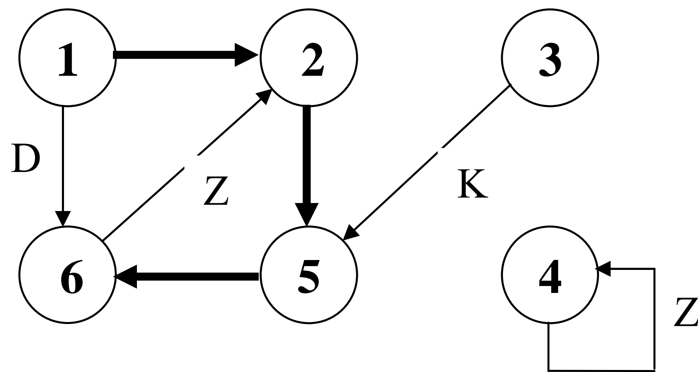
Klasifikace hran

1. *Stromové hrany.* Jsou hrany patřícím stromům lesa. Hrana ***(u,v)*** je stromová byl-li vrchol ***v*** objeven z vrcholu ***u***.

2. *Zpětné hrany*. Jsou hrany ***(u,v)**,* kde vrchol ***v*** je předkem vrcholu ***u***, tj. vrchol ***v*** leží na cestě z kořene DFS stromu do vrcholu ***u**.* Hrana *(u,u)* je považována za zpětnou.

3. *Dopředné hrany*. Jsou hrany *(u,v),* kde vrchol ***u*** je předkem vrcholu ***v*** v DFS stromě.

4. *Křižující hrany.* Všechny ostatní hrany. Spojují vrcholy jednoho DSF stromu pokud jeden z nich není předkem druhého anebo vrcholy různých stromů lesa DSF stromů.



Když hranou poprvé procházíme, potom pro hrana *(u,v)* je:

1. Je-li vrchol ***v*** bílý, **stromová**.

2. Je-li vrchol ***v*** šedý, **zpětná**.

3. Je-li vrchol ***v*** černý a ***objeven(u) < objeven(v),*** **dopřední**.

4. Je-li vrchol ***v*** černý a ***objeven(u) > objeven(v),*** **křižující**.

V případě *neorientovaného* grafu je ***(u,v)*** a ***(v,u)*** tatáž hrana.

Hrana je potom klasifikována podle prvního z jejich vyjádření, které nalezne algoritmus prohledávání do hloubky.

Je možné ukázat, že při prohledávání neorientovaného grafu do hloubky, každá z jeho hran je buď stromová nebo zpětná.

Zpětné hrany jsou klíčem, k nalezení cyklů v grafu.

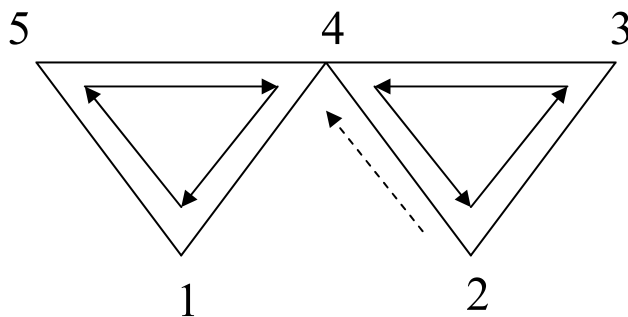
Vznikne-li prohledáním grafu do hloubky zpětná hrana, graf obsahuje cyklus. Je-li ***(u,v)*** zpětná hrana, potom cesta z ***v*** do ***u*** a tato hrana tvoří cyklus.

Obráceně, lze dokázat, že je-li v grafu cyklus, jeho prohledáním do hloubky musí vzniknout zpětná hrana.

Příklad – Eulerův cyklus

1. Ověř existenci Eulerova cyklu v grafu

2. Spoj hranově disjunktní cykly



Eulerův cyklus – 2 3 4 1 5 4 2

- hrany nalezených cyklů odstráníme a vrcholy cyklu vložíme do zásobníku
- nejsou-li odstraněny všechny hrany, vybereme vrcholy až k vrcholu, z kterého vychází hrana

zásobník

2
2 3
2 3 4
2 3 4 2 → <u>2</u>
2 3 4
2 3 4 1
2 3 4 1 5
<u>2 3 4 1 5 4</u>

Topologické řazení

Máme množinu prvků, ve které je definováno uspořádaní pro některé dvojice ***(u,v)***

– např. prvky jsou činnosti v čase (***u*** předchází ***v***)

Uspořádaná dvojice prvků ***(u,v)*** potom tvoří hranu orientovaného acyklického grafu (*directed acyclic graph* – *DAG*)

Algoritmus topologického řazení

Úkol: Lineárně seřadit vrcholy (činnosti) tak, aby
vzájemné pořadí dvojic zůstalo zachováno

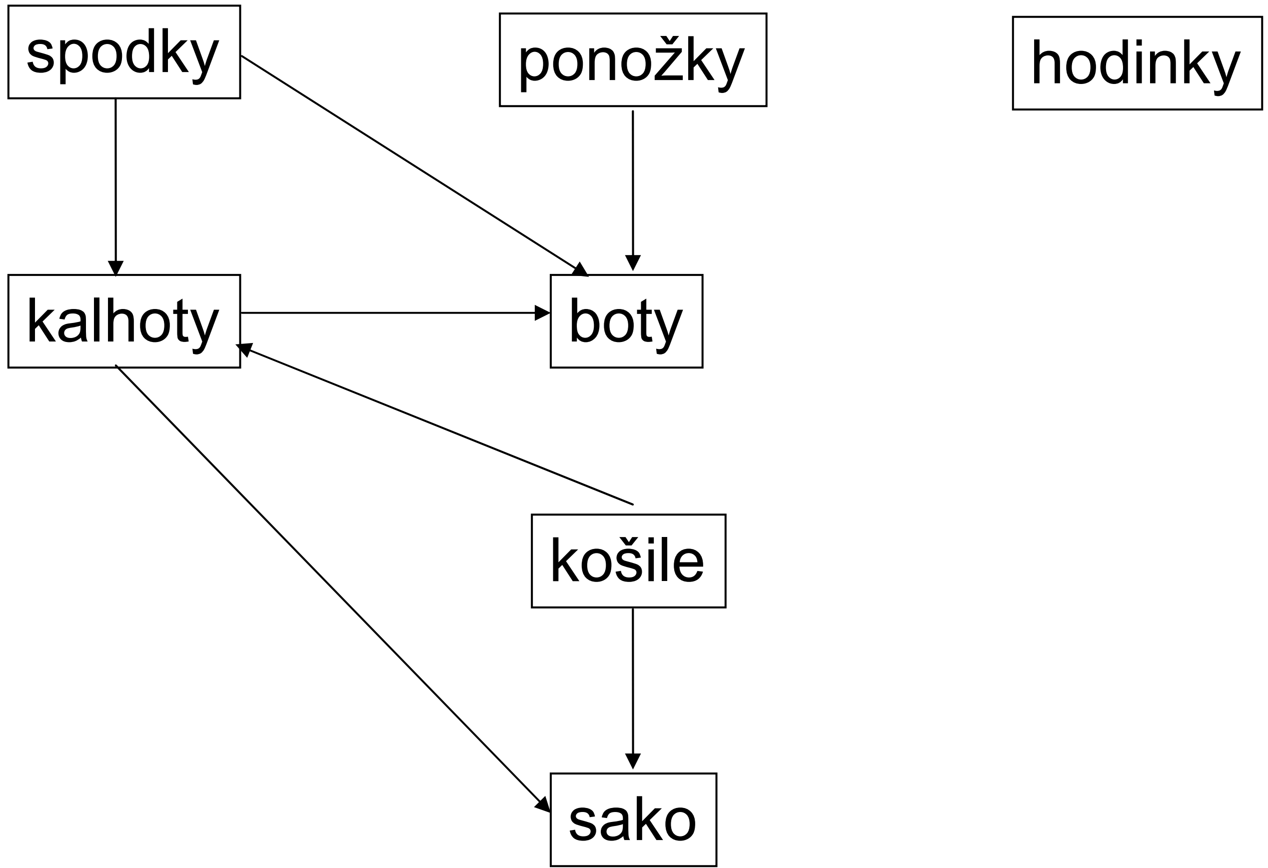
Algoritmus:

- Zavolej DFS a vytvoř les DFS stromů.
- Když je vrchol ukončen přidej ho do na začátek seznamu
- Vrať seznam vrcholů

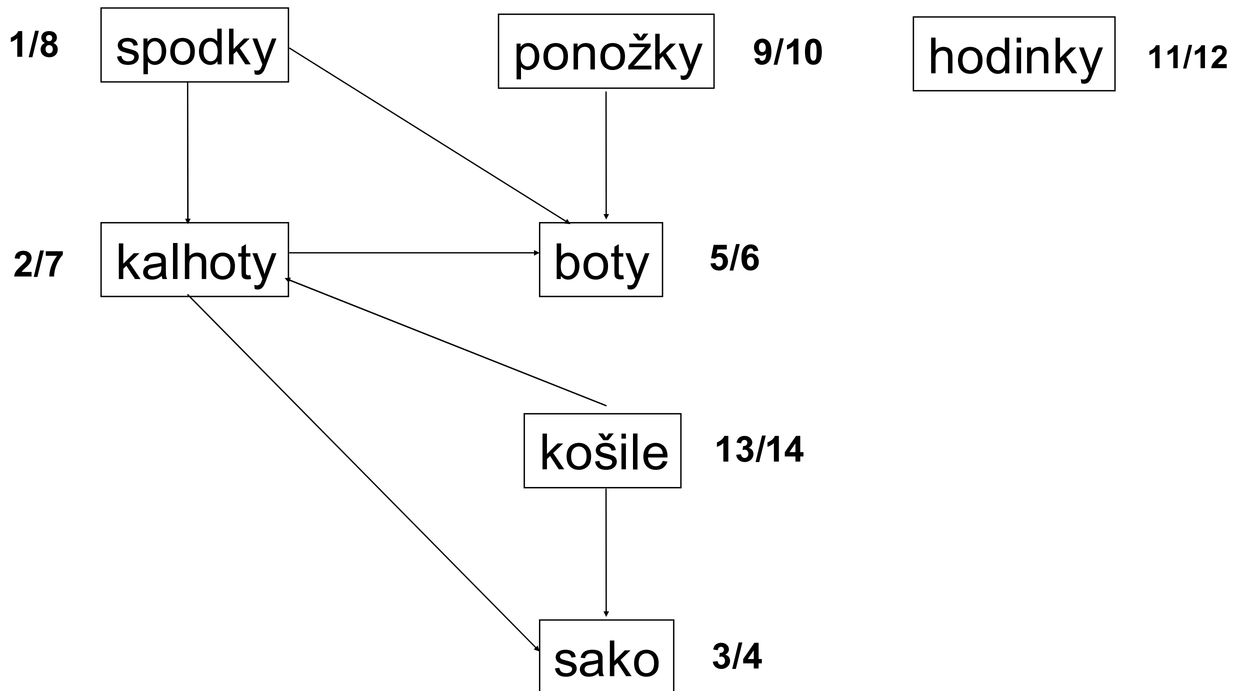
Čas výpočtu topologického řazení je ***O(|V | + |H|).***

Příklad – oblékání

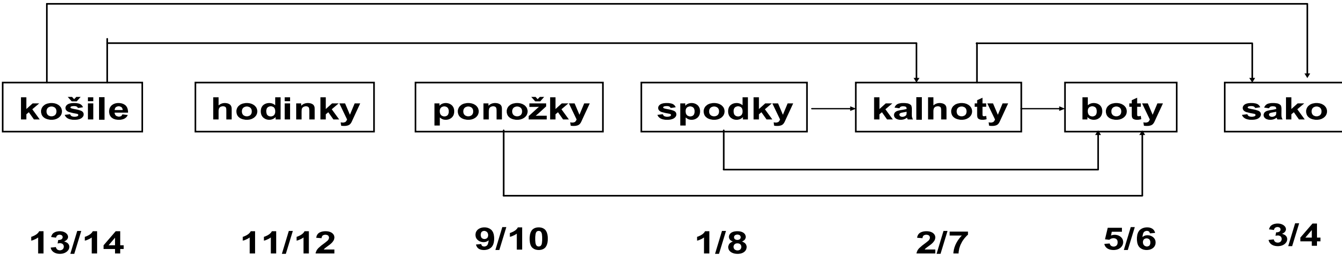
Vrcholy grafu budou tvořeny oblékanými součástmi garderobiéry.
Uspořádaní dvojic vrcholů je dáno požadovaným pořadím oblékání.
Tyto dvojice definují hrany acyklického orientovaného grafu.


spodky
ponožky
hodinky
kalhoty
boty
košile
sako

Po vykonání DFS získáme například hodnoty časů *objevení/dokončení* jednotlivých vrcholů na následujícím obrázku.



seřazení vrcholů



Test